STAFSEA^2

STructured **A**daptive **F**low **S**olver for
Engineering and **A**tmospheric **A**pplications

User's Guide

Matthias Waidmann

Geophysical Fluid Dynamics
Rupert Klein

Institut of Mathematics
Department for Mathematics and Informatics

Free University, Berlin, Germany

Version 0.1

XXXXXX 2014

ABSTRACT

CONTENTS

Abstract i

Contents iii

List of Tables iv

List of Figures v

List of Abbreviations vii

List of Symbols viii

1. Introduction 1

2. Governing Equations 2

3. Framework Overview 3

4. Framework Installation 6

4.1 GCC 6

4.2 MPICH 6

4.3 ZLIB 7

4.4 HDF5 7

4.5 Silo 7

4.6 HYPRE 8

4.7 SAMRAI 8

4.8 VisIt 11

4.9 Environmental Variables 11

4.10 STAFSEA^2 12

5. STAFSEA^2 Usage 14
5.1 Compile Setup 14
5.2 Computation Setup 17
5.3 Execution 20
5.4 Data Output and Visualization 21
5.4.1 Field Data 22
5.4.2 Global Data 26
5.4.2.1 *GlobalQuantitiesMonitor.m* for reading *globalQuantitiesTimeSeries.inf* 26
5.4.2.2 *TimeStepMonitor.m* for reading *timeSteps.inf* . . . 28
5.4.2.3 *InitialPressureErrorMonitor.m* for reading *initialPressureErrorMeasure.inf* 29
5.5 Initial Conditions 30

6. Convergence Test Toolkit (CTT) 31
6.1 Convergence Test Configuration 33
6.2 CTT Description 36
6.3 Output Files 48
6.3.1 `errorNorms.dat` 48
6.3.2 `convergenceOrder.dat` 48
6.3.3 `variables.inf` 48
6.4 Visualization 49
6.4.1 Visualization using the `convOrdViz` script 49
6.4.2 Visualization of spatial error distribution and location of maximum error using VisIt 49
6.5 CTT Limitations 50

7. Solver Test Environment (STE) 51

LIST OF TABLES

4.1 Successfully operated combinations of package versions 13

5.1 Basic modes the code can be operated in 17

5.2 Field data for visualization with VisIt 22

5.3 Basic Information 23

5.4 Additional field data specified in VisIt 23

5.5 Additional field data in debug mode for visualization using VisIt . . 25

5.6 Global data for code monitoring at time level $n + 1$; *fourth column*: $\otimes$: exactly conserved, $\oslash$: discretely conserved, $\ominus$: not conserved; *fifth column*; code modes A,B,C and D are specified in table 5.1 . . 27

LIST OF FIGURES

6.1 Hierarchy of grids (a) with and (b) without fine grid for reference solution 33

6.2 Matching of solutions on congruent patches (fat) of grids of different resolution based on patch coordinates (not patch numbers); here a refinement factor $r = 2$ is used between the two grids per coordinate direction and each domain is divided into 4 sub-domains (boxes/patches) 0-3 according to 4 processors used for parallel solution generation. The box/patch layout is identical on each grid. . . . 41

6.3 Grids with (a) arbitrary refinement factor $r \in \mathbb{R}^+ \backslash \mathbb{N}$ and (b) $r \in \mathbb{N}$ (here $r = 2$) 47

LIST OF ABBREVIATIONS

LIST OF SYMBOLS

1. INTRODUCTION

XXXXXXXXXXXX

2. GOVERNING EQUATIONS

XXXXXXXXXXXXXX

3. FRAMEWORK OVERVIEW

The presented numerical method is implemented as SAMRAI[1] based parallel C++ program with adaptive mesh refinement capabilities. It requires the third party open source software listed below. Therein, the version numbers given in brackets represent the respective version of the third party software used for the computations within this thesis. More recent versions – if available – can be tested, however, functionalities and/or software interfaces might have changed, potentially leading to malfunction of the entire framework. The footnotes point to the websites the software is available from. These websites not only offer access for downloading the respective open source third party packages but also give detailed overview over the respective software including citations of relevant literature. Partially, the packages might also be available from the package manager of the machine the code framework should be installed on (in this case this it the preferred installation option), especially the ones listed as "basic software packages" in the following list of required software:

- Basic Software Packages:
  - **CLI**: A command-line interface or terminal is needed for installation of the other third party software packages as well as compilation and execution of the STAFSEA[2]. Commonly it is a utility of the operating system of the machine the framework should be installed and executed on.
  - **GCC** (4.5.4)[2]: The GNU Compiler Collection (GCC) or any other compiler collection containing C, C++ and Fortran compilers is both needed for installation of the other third party software and for generation of an executable of the implementation of the presented numerical method.
  - **MPICH** (1.3)[3]: A message passing interface (MPI) is required for data exchange during parallel computation on more than one processor.

[1] see page 5
[2] `http://gcc.gnu.org/`
[last visited on June 24^{th}, 2014]
[3] `http://www.mpich.org/`
[last visited on June 24^{th}, 2014]

  - **ZLIB** (1.2.8)[4]: ZLIB is a free library for lossless data compression and it is used by HDF5 and SAMRAI listed below.
  - **Doxygen** (1.8.5)[5]: Doxygen is a tool for generation of documentation from source code. It supports many programming languages such as C, C++ or Fortran. It is needed for creation of an .html and/or LaTeX/.pdf documentation of both the SAMRAI library and STAFSEA[2] and has to be available during installation of SAMRAI.
  - **HDF5** (1.8.8)[6] : The Hierarchical Data Format 5 (HDF5) is a technology for managing large and complex data. It is the basis for the adaptively refined parallel output and restart data of SAMRAI.

- Software Packages by Lawrence Livermore National Laboratory (LLNL)[7]:
  - **Silo** (4.8-bsd)[8]: The Silo library implements an application programing interface (API) which is designed for reading and writing of scientific data (e.g. in HDF5 format) to binary disk files.
  - **HYPRE** (2.9.0b)[9]: The high performance preconditioner library HYPRE features scalable parallel multigrid methods for solving large, sparse, linear systems of equations on parallel computers on both structured and unstructured grids. It is used as core solver for each linear system arising in the correction steps of the present numerical method due to global – or, at least, non-local – coupling in space. Therefore the Semi-Structured-Grid System Interface (SStruct) is used as it is able to handle, for example, block structured or adaptively refined structured meshes. It is composed out of structured grid parts with stencil based graphs establishing the relationship between part data. XXXXXX In multigrid methods solutions on coarser grids are used for damping of errors on finer grids.

[4] `http://www.zlib.net`
[last visited on June 24^{th}, 2014]
[5] `http://www.stack.nl/~dimitri/doxygen/`
[last visited on June 24^{th}, 2014]
[6] `http://www.hdfgroup.org/HDF5/`
[last visited on June 24^{th}, 2014]
[7] `https://www.llnl.gov/`
[last visited on June 24^{th}, 2014]
[8] `https://wci.llnl.gov/codes/silo/`
[last visited on June 24^{th}, 2014]
[9] `http://computation.llnl.gov/casc/linear_solvers/sls_hypre.html`
[last visited on June 24^{th}, 2014]

- **SAMRAI** (3.1.0-beta)[10]: The Structured Adaptive Mesh Refinement Application Infrastructure (SAMRAI) library serves as code basis and its functionalities are utilizes for:
  - grid management
  - data management
  - data output
  - computation restart
  - parallel computation based on domain decomposition including load balancing
  - adaptive mesh refinement (AMR) for local flow and/or geometry feature based grid refinement

  Therefore the computational grid is organized in a hierarchy of grid levels with the coarsest level covering the entire computational domain and the finer levels addressing only sub-domains of increased interest. Each grid level is sub-divided into boxes containing patches which can be distributed to different processors for parallel computation via the included load balancer. On each single patch the kernels of the numerical method can be implemented as for single processor computation while the data on different patches is connected via MPI communication using ghost cell layers beyond the actual patch extent. The data is stored via Silo in the HDF5 based .samrai format which can be read by VisIt via a .VisIt file listing the different data sets over time.
- **VisIt** (2.6.3)[11]: VisIt is an interactive open source visualization and analysis tool which can read the SAMRAI data output in .samrai format by default and in parallel if required. For visualization of the time series of field data the .VisIt file, generated by SAMRAI during code execution, has to be opened. It lists the points in time at which data sets each of which could have been generated distributed to different processes are available.

The listed tools allow for compilation and parallel execution of STAFSEA[2] on adaptively refined meshes with suitable management, storage and visualization of the generated data.

[10] `https://computation-rnd.llnl.gov/SAMRAI/`
[last visited on June 24^{th}, 2014]

[11] `https://wci.llnl.gov/codes/visit/`
[last visited on June 24^{th}, 2014]

4. FRAMEWORK INSTALLATION

In this section installation of the third party framework listed in the previous section is described. The table on page 13 gives an overview over combinations of package versions which have been installed and run successfully. The installation procedure has to be done according to the schedule given in this section for the whole framework to work properly. Each installation process basically consists of the command-line building commands

```
./configure
```

```
make -j <number of processors to be used>
```

and the installation

```
make install
```

if not stated otherwise below. The expression <*number of processors to be used*> represents the number of processors that should be used for the compiling and linking process in order to speed it up.

4.1 GCC

If a C/C++/Fortran compiler collection is not yet available on the machine the framework should be installed on, get and install GCC from the package manager of the machine or from the website given on page 3.

4.2 MPICH

If MPI is not yet available on the machine the framework should be installed on, install MPICH either via the package manager or manually after downloading from

the respective website as given on page 3. The `mpirun` command has to be available. If it is not contained in the folder the other MPI executables (such as mpiexec, mlicc ....) XXXXXXXX

4.3 ZLIB

If the ZLIB library is not yet available on the machine the framework should be installed on, install ZLIB either via the package manager or manually after downloading from the respective website as given on page 4.

4.4 HDF5

The HDF5 package is recommended to configure with the configure options

```
./configure -prefix=<hdf5TargetPath>
-enable-parallel -disable-fortran -with-zlib=<path>
CC=<mpiPath>/bin/mpicc CXX=<mpiPath>/bin/mpicxx
FC=<mpiPath>/bin/mpif90 F77=<mpiPath>/bin/mpif77
```

before compiling and linking (`make`) as well as installation (`make install`). Italic expression framed by <...> represent the respective paths to directories the packages should be installed to (prefix) or installed packages can be found in. Especially, the placeholder <*path*> points to the directory the folders `bin`, `include` and `lib` can be found in, such as `/usr` or `/opt/local`, as ZLIB is usually located in the `lib` folder as `libz` library file.

4.5 Silo

XXXXXX

```
./configure -prefix=<siloTargetPath>
-with-hdf5=<hdf5TargetPath>/include,<hdf5TargetPath>/lib
-with-zlib=<path>/include,<path>/lib -disable-fortran
CC=<mpiPath>/bin/mpicc CXX=<mpiPath>/bin/mpicxx
FC=<mpiPath>/bin/mpif90 F77=<mpiPath>/bin/mpif77
```

4.6 HYPRE

XXXXXXXX

```
./configure -prefix=<hypreTargetPath> -disable-fortran
-with-MPI-include=<mpiPath>/include/<mpichName>
-with-MPI-libs=mpich
-with-MPI-lib-dirs=<mpiPath>/lib/<mpichName>
CC=<mpiPath>/bin/mpicc CXX=<mpiPath>/bin/mpicxx
F77=<mpiPath>/bin/mpif77
```

4.7 SAMRAI

If `SAMRAI v3.1.0-beta` is used (which is recommended as it is the SAMRAI version STAFSEA[2] was developed with), a few issues have to be resolved before the library can be build and installed:

In file

```
<samraiPath>/source/SAMRAI/hier/PatchHierarchy.C
```

1) the code block on page 10 has to be added after line 921 of the original file. Line 919 - 921 read

```
getFromDatabase(database,
VariableDatabase::getDatabase()
->getPatchDataRestartTable() );
```

2) Lines 829 - 831 of the original file reading

```
database->putIntegerArray("d_proper_nesting_buffer",
&d_proper_nesting_buffer[0],
static_cast<int>(d_proper_nesting_buffer.size()) );
```

have to be changed to

```
if (d_max_levels > 1)
{
database->putIntegerArray("d_proper_nesting_buffer",
&d_proper_nesting_buffer[0],
static_cast<int>(d_proper_nesting_buffer.size()) );
}
```

3) the two lines

```
if (d_max_levels == 1)
d_proper_nesting_buffer.push_back(1);
```

have to be added after line 212

```
d_proper_nesting_buffer.clear();
```

of the original file and before the line

```
for ( int ln=0; ln<d_max_levels-1; ++ln ) {
```

Further, in file

```
<samraiPath>/source/SAMRAI/tbox/RestartManager.C
```

the line

```
#include <iostream>
```

has to be added at the very top of the file before the line

```
#include <string>
```

Note, that newer SAMRAI versions 3.X.Y with X>1 might not be compatible with STAFSEA[2] by default since interfaces might have changed.

```
d_connectors_to_domain.resize(d_number_levels,
tbox::Pointer<Connector>(NULL) );
for ( int ln=0; ln<d_max_levels; ++ln )
{
d_patch_levels[ln]
->getMappedBoxLevel()->cacheGlobalReducedData();

/*
* Set d_connectors_to_domain[ln] by either bridging the
* new mapped _box_level across the coarse
* mapped_box_level or using Connector::findOverlaps.
* Using findOverlaps is probably faster because the
* domain mapped_box_level usually has a fixed number of
* boxes and is already in a globalized state.
*
* d_connectors_to_domain has the samg gcw as the peer
* connectors (peer gcws are larger than gcw to finer).
* This is an arbitrary decision which can be reviewed
* later.
*/

d_connectors_to_domain[ln] = new Connector(
*d_patch_levels[ln]->getMappedBoxLevel(),
d_domain_mapped_box_level,
getRequiredConnectorWidth(ln, ln),
MappedBoxLevel::DISTRIBUTED);

OverlapConnectorAlgorithm oca;
oca.findOverlaps(*d_connectors_to_domain[ln]);
}
```

After fixing this issue the recommended configuration command is

```
./configure -prefix=<samraiPath>/objs
-with-CC=<mpiPath>/bin/mpicc
-with-CXX=<mpiPath>/bin/mpicxx
-with-F77=<mpiPath>/bin/mpif77 -with-x
-with-mpi-include=<mpiPath>/include/<mpichName>
-with-mpi-libs=mpich
-with-mpi-lib-dirs=<mpiPath>/lib/<mpichName>
-with-MPICC=<mpiPath>/bin/mpicc
-with-doxygen=<doxygenPath> -with-hdf5=<hdf5TargetPath>
-with-silo=<siloTargetPath>
-with-hypre=<hypreTargetPath> -with-zlib=<path>
CC=<mpiPath>/bin/mpicc CXX=<mpiPath>/bin/mpicxx
F77=<mpiPath>/bin/mpif77 CXXFLAGS=<compileOption>
CFLAGS=<compileOption> FFLAGS=<compileOption>
```

where <*compileOption*> is either `-g` for debug mode or `-O2` or `-O3` for optimized code for high performance computing. More options can be checked by typing

```
./configure -help
```

4.8 VisIt

Besides of the source code VisIt can be downloaded (see footnote on page 5) as executable for various platforms which is the recommended option.

4.9 Environmental Variables

The `Makefile` of STAFSEA^2 includes the SAMRAI `Makefile.config` and the XXXXXX file `Makefile.para` for selection of compile options. The latter uses the defined environment variables

- `SAMRAIDIR`
- `SAMRAIDIRobjs`

for making the SAMRAI library available to STAFSEA^2. The environment variables should be set “permanently”. If a `bash` shell is used as CLI on a UNIX

based system (Unix/Linux/MAC OSX), the environment variables are set in the file `${HOME}/.bashrc` by adding

```
export SAMRAI="<samraiPath>"
export SAMRAIDIR="${SAMRAI}"
export SAMRAIDIRobjs="${SAMRAI}/objs"
```

at the end of the file, for example. After the modification is saved, each time a shell is opened (or after typing `/bin/bash` in the command-line to restart the shell) the specified environment variables are set and ready for use by the linker during the building process of STAFSEA^2.

4.10 STAFSEA^2

STAFSEA^2 does not need to be installed. If the executable XXXXXX is already available and its building has been done with the desired parameters in the compiler flag file `Makefile.para`, it can be executed as described in section 5.3. If compiler flags in `Makefile.para` need to be changed, the existing executable and related object files (`.o`) can be removed by the

```
make clean
```

command. After setting the desired compile flags in `Makefile.para` (see "User's Guide" below) new object files and a new executable including linked SAMRAI library routines are generated via execution of the command-line command

```
make -j <numberProcesses>
```

in the main code directory (the one the file `Makefile.para` is located in). After the building process including compiling and linking is completed the executable is ready to be run as described above.

Tab. 4.1: Successfully operated combinations of package versions

Operating System	Compiler	MPI	HDF5	Silo	hypre	SAMRAI
MAC OSX 10.8.5	gcc 4.5.4	mpich-gcc45 3.1.2	1.8.8	4.8-bsd	2.8.0b (fixed)	3.1.0-beta (fixed)
MAC OSX 10.6.8	gcc 4.5.3	mpich2 1.4.1p1	1.8.8	4.8-bsd	2.8.0b (fixed)	3.1.0-beta (fixed)
Linux Fedora 15	gcc 4.6.3	mpich2 1.4.1p1	1.8.7	4.8-bsd	2.8.0b (fixed)	3.1.0-beta (fixed)
Ubuntu 14.4 / CentOs 7	gcc 4.8.2	mpich2 1.4.1p1	1.8.8	4.8-bsd	2.8.0b (fixed)	3.1.0-beta (fixed)
Xubuntu 14.04.01 LTS	gcc 4.8.2	mpich2 1.2.1	1.8.4	4.6.2	2.8.0b (fixed)	3.1.0-beta (fixed)
Linux Debian			1.8.12	4.8-bsd	2.8.0b (fixed)	3.1.0-beta (fixed)
SuSE Linux (SLES)	intel 11.0.083	mvapich2 1.7a2-intel	1.8.6-mvapich2-intel	4.8-bsd	2.8.0b (fixed)	3.1.0-beta (fixed)

5. STAFSEA[2] USAGE

This section describes how the whole framework is used once installed successfully according to section 4. The two relevant files besides of the code executable XXXXXXXX are

- the compile parameter file `Makefile.para` in the main code directory, described in the following section "Compile Setup" (5.1)
- the runtime parameter file <*InputFileName*><*X*>`d.input`[1] in the directory `./InputFiles`, described in the subsequent section "Computation Setup" (5.2)

5.1 Compile Setup

The compile setup can be changed in the file `Makefile.para`. After changing one or more of the following compiler flags contained in this file, the code executable has to be built again as described in section 4.10. These options are available:

- `COMPILEMODE`:
  - debug mode: `-g`
  - optimized[2]: `-O`<*n*>

    Note: also the third party packages listed in section 3, especially SAMRAI and HYPRE, have to be compiled with optimized flags for high performance computing during the installation procedure described in section 4.

[1] "<*X*> represents the spatial dimension of the computational problem and, thus, is either "2" or "3"

[2] for specification of <*n*> see e.g. `http://gcc.gnu.org/onlinedocs/gcc-3.2/gcc/Optimize-Options.html` [last visited on Aug 13^{th}, 2014]

- CPPFLAGS_EXTRA:
  extra options for compiling the code specifically adapted to the problem that has to be solved:
  - NDIM:
    switches between the two dimensional (NDIM = 2) and the three dimensional (NDIM = 3) version of the code
  - PHAS:
    switches between single-phase (PHAS = 1) and two-phase (PHAS = 2) flow solver; table 5.1 lists all possible options in combination with the LESE parameter
  - ORD:
    switches between first (ORD = 1) and the second (ORD = 2) order accuracy
  - FRIC:
    switches the friction operator on (FRIC = 1) or off (FRIC = 0)
  - DIFF:
    switches the diffusion operator on (DIFF = 1) or off (DIFF = 0)
  - IMP:
    predictor mode:
    * IMP = 0: a fully explicit predictor including explicit treatment of the friction operator if switched on (FRIC = 1)
    * IMP = 1: a semi-implicit predictor: advection is treated explicitly and both friction and diffusion are treated implicitly (if switched on) with the implicit part in the center of the second order operator splitting (if ORD > 1)
    * IMP = 2: a semi-implicit predictor: advection is treated explicitly and both friction and diffusion are treated implicitly (if switched on) with the explicit part in the center of the second order operator splitting (if ORD > 1)

    the species diffusion operator on (DIFF = 1) or off (DIFF = 0)
  - LIM:
    specifies the second order TVD slope limiter which is used for state reconstruction during flux computation in the explicit predictor part
    * LIM = 0: no limiter (central difference)

  * LIM = 1: minmod [**?**]
  * LIM = 2: monotonized central [**?**]
  * LIM = 3: Osher [**?**]
  * LIM = 4: Sweby [**?**]
  * LIM = 5: superbee [**?**]
  * LIM = 6: van Leer [**?**]
  * LIM = 7: van Leer - Klein [**?**]
  * LIM = 8: Koren [**?**]
  * LIM = 9: UMIST [**?**]
  * LIM = 10: van Albada 1 [**?**]
  * LIM = 11: ospre [**?**]

- NOPR:
  specifies if the second projection step is a nodal (NOPR = 1) or a cell centered (NOPR = 0) projection
- EXPR:
  switches between exact (EXPR = 1) and approximate (EXPR = 0) second projection in case of nodal second projection (NOPR = 1)
- LESE:
  switches level-set module on (LESE = 1) and off (LESE = 0); table 5.1 lists all possible options in combination with the PHAS parameter
- OUT:
  decides which parts of the code are switched on
  * OUT = 0: entire predictor-corrector scheme
  * OUT = 1: predictor only
  * OUT = 2: corrector only
  * OUT = 3: predictor and first projection of corrector step
- DISP:
  switches between different (screen) output modes:
  * DISP = 0: (almost) no output for high performance computing
  * DISP = 1: default output
  * DISP = 2: verbose output for debugging purposes

Tab. 5.1: Basic modes the code can be operated in

mode	PHAS	LESE	description
A	0	1	level-set only, advected in a fix velocity field
B	1	0	single phase flow solver
C	1	1	single phase flow solver including advected level-set
D	2	<*arbitrary*>	two-phase flow solver with sharp level-set interface

5.2 Computation Setup

In the input file <*InputFileName*><*X*>`d.input`[3] in the folder `./InputFiles` located in the main code directory computation specific parameters are set which can be modified without recompiling the code. The input file is divided into different sections with respect to the usage of the parameters in the program. These sections are available:

- `CartesianGeometry`:
  in this section the (initial) Cartesian grid resolution on the coarsest grid level is specified via the computational domain size (`x_lo`: coordinates of lower left (foreground) corner, `x_hi`: coordinates of upper right (background) corner) and the indices of the lower left (foreground) and the upper right (background) grid cell (`domain_boxes`).

- `DomainBoundary`:
  specifies the type of each domain boundary. Available types are `"wall"`, `"slipWall"`, `"inflow"`, `"outflow"` and`"periodic"`.

- `EquationSetUp`:
  number, names, and location (`"cell"`, `"node"`) of conserved and additional variables can be specified in this section.

[3] see footnote 1 on page 14

- `FlowSolver`: computation and fluid related parameters:
  - initial time `t_beg` in seconds, final time `t_end` in seconds, largest allowable time step size `dt_max` in seconds, CFL number `CFL`, maximum number of time steps `max_timesteps`, maximum data output interval `write_steps` in time steps and maximum time `write_period` to next data output in seconds
  - gravitational acceleration `g` in $\frac{m}{s^2}$
  - inflow parameters (density `rho_inflow`, velocity `velocity_inflow`, flag for selection of velocity profile `inflow_profile`)
  - reference values (density `rho_ref` in $\frac{kg}{m^3}$, pressure `p_ref` in $\frac{kg}{s^2 m}$, temperature `T_ref` in K, velocity `velocity_ref` in $\frac{m}{s}$
  - fluid properties
  - flags to choose
    - initial data setup `iniCondIdx` XXXXXXXXXX
    - if the velocity field should stay fixed throughout the computation (`velocity_field_fix`)
    - whether the initial pressure $p^{(2)}$ is set or computed (`prescribe_initial_pressure`)
    - whether global data (section 5.4.2) is recorded (`monitor_global_quantities`)
  - tuning parameters for
    - state initialization (maximum number of iterations if initial pressure is not set (`iniIterMaxNo`), threshold for iteration termination (`iniIterTol`)[4] and error measure `iniIterMode` for initial pressure determination (0: L_∞-norm, 1: L_1-norm, 2: L_2-norm))
    - relative and absolute Poisson solver tolerances
- `GriddingAlgorithm`:
  tuning parameters for setting up and adapting the grid hierarchy
- `LevelSet`:
  input parameters for the level-set module such as grid level the level-set function is advected on (`levelset_level`), width of the narrow band around the

[4] 100 `iniIterTol` % deviation of a chosen measure of the initial pressure $p^{(2)}$ between iteration step $\mathcal{I}$ and $\mathcal{I}-1$

level-set function zero level (narrow_band_width), level-set initial conditions setup and reinitialization control

- LoadBalancer:
  parameters for the load balancer for parallel computation

- PatchHierarchy:
  in this section general parameters of the AMR grid such as number of grid levels (max_levels), grid cell size ration (ratio_to_coarser) between subsequent levels, minimum number of grid cells on grid level $\mathcal{L}$ between level $\mathcal{L}-1$ and $\mathcal{L}+1$ (proper_nesting_buffer) and both smallest (smallest_patch_size) and largest (largest_patch_size) patch size on a grid level are set

- Restart:
  in this section the name of the folder (restart_write_dir) to write restart data to and the interval (restart_interval) after which restart data is written (in time steps) are selected. Further, it is specified whether older checkpoints are kept or deleted after a more recent one is available in order to save disk space (keep_only_latest_file) and whether previously written files for global data and files for global data written after restart are assembled to one resulting file for global data (previous_output_available). In the folder ${SAMRAIDIR}/docs/userdocs a separate document on restart can be found.

- StandardTagAndInitialize:
  definition of grid refinement method (tagging_method) and initial grid level configuration (Refine_Boxes) via specification of the initially refined areas on each grid level

- TimerManager:
  in this section parameters for measurement of time consumption of the different code sections are set. A separate document on timing in SAMRAI can be found in the folder ${SAMRAIDIR}/docs/userdocs.

- TwoPhaseFlow:
  in this section additional fluid properties are specified in case of two-phase flow

5.3 Execution

Execution of the compiled code can be started from the main code directory. It is the one the executable named <*executableName*> (e.g. `flowSolver`), the file `main.C` and the folder `./InputFiles` are located in. Input files in this folder are named <*InputFileName*><*X*>`d.input`[5]. They contain prescribed parameters specific for the respective computational problem. The command-line command

```
mpirun -np <numberProcesses> ./<executableName>
./InputFiles/<inputFileName><X>d.input
```

initiates a new computation using <*numberProcesses*> processes while the command

```
mpirun -np <numberProcesses> ./<executableName>
./InputFiles/<inputFileName><X>d.input ./<RestartData>
<storedRestartDataTimeStepNumber>
```

launches a computation based on given restart data in the <*RestartData*> folder. All time step numbers <*storedRestartDataTimeStepNumber*>, at which restart data are available, can be found via

```
ls ./<RestartData>
```

taking the numbers appended to the `restore` folders in the displayed list with or without leading zeros. The number of processes <*numberProcesses*> to be used for the computation ideally equals the number of available processor cores in order to avoid core-internal communication. For minimizing time consuming communication between different processes, the allowed patch size should be adjusted in the input file such that each process operates on as little patches as possible. If a different number of processes is used at restart as has been used for generation of the restart data, the latter first have to be reorganized for being usable with the desired amount of processes. This transformation can be done with the command sequence

[5] see footnote 1 on page 14

```
mkdir ./RestartDataTransformed

${SAMRAIDIRobjs}/bin/restart-redistribute
./<RestartData> ./RestartDataTransformed
<storedRestartDataTimeStepNumber>
<numberDesiredProcesses>

mv ./<RestartData>/*.inf ./RestartDataTransformed

mv ./<RestartData>/dumps.visit ./RestartDataTransformed

rm -rf ./<RestartData>

mv ./RestartDataTransformed ./<RestartData>
```

using the SAMRAI executable `restart-redistribute` before code restart. The environmental variable `${SAMRAIDIRobjs}` is the one set in section 4.9.

When the code starts running, the file `do.inf` is generated and after successful code termination the file `done.inf` is written. These files can be used by external scripts to automatically determine if the code has started and/or terminated successfully.

5.4 Data Output and Visualization

Two types of data are produced by the code:

- **field data**[6] for visualization in VisIt (see page 5)
- **global data** for visualization using MATLAB®[7] or GNU Octave[8].

[6] Note that field data for visualization is single precision in order to reduce the required disk space. Computations based on intermediate data therefore always have to be done based on the double precision restart data in the *./<RestartData>* folder.

[7] MATLAB® is a high-level interpreted language and interactive environment for numerical computations, visualization and programming by MathWorks:
`http://www.mathworks.de/products/matlab/`
[last visited on August 01st, 2014]

[8] GNU Octave is an open source high-level interpreted language for numerical computations:
`http://www.gnu.org/software/octave/`
[last visited on August 01st, 2014]

5.4.1 Field Data

Field data can be accessed by opening the file `dumps.visit` in VisIt (*File → Open file ...*). The file is located in the `VisItOutput` folder in the main code directory and lists the time steps with available field data. Single field quantities can be displayed by adding a variable with suitable display mode via the VisIt *Add* button. The following table 5.2 lists the available data written by the code, in which the first column specifies the variable, the second lists the corresponding variable names in VisIt, the third column indicates the locations of the quantities (c = cell center, n = node), the fourth column reports if the respective quantity is a scalar (1) or a vector (2,3) with 2 and 3 referring to the number of spatial dimensions, the fifth column displays in which mode of the code the quantity is available (see table 5.1 for the different available code modes) and the sixth column recommends the preferred display mode to choose in VisIt.

Tab. 5.2: Field data for visualization with VisIt

1	2	3	4	5	6
var.	**VisIt var.**	**loc.**	**dim.**	**avail.**	**recomm. display mode**
$\overline{P}$	energy	c	1	B,C,D	Pseudocolor, Contour
$\overline{\rho}$	density	c	1	B,C,D	Pseudocolor, Contour
$\overline{\rho \boldsymbol{v}}$	momentum	c	2,3	B,C,D	Vector
$\lvert\overline{\rho \boldsymbol{v}}\rvert$	momentum_mag.	c	1	B,C,D	Pseudocolor
$\boldsymbol{v}$	velocity	c	2,3	A,B,C,D	Vector
$\lvert\boldsymbol{v}\rvert$	velocity_mag.	c	1	A,B,C,D	Pseudocolor
$p^{(2)}$	pressure	n	1	B,C,D	Pseudocolor, Contour
$-$	center-increment	c	1	B,C,D	Pseudocolor, Contour
$\overline{P\varphi}$	VoF	c	1	D	Pseudocolor
$\mathcal{N}$	narrow band	c	1	A,D	Pseudocolor
$-$	cut cell marker	c	1	A,D	Pseudocolor
G	level-set, cell	c	1	A,D	Pseudocolor, Contour
G	level-set, node	n	1	A,D	Contour

The following information is provided by VisIt based on the code output by default:

Tab. 5.3: Basic Information

VisIt var.	**description**	**recomm. display option**
amr_mesh	grid cell boundaries	Mesh
levels	levels in the AMR hierarchy	Subset
patches	domain decomposition	Subset

Additional quantities can be derived In VisIt (*Controls* → *Expressions* ...) based on the ones provided by the code data output. A set of additional quantities has been specified which is stored in the file `AdditionalQuantities.xml` in the `./Utilities/VisItSettings` folder. This set can be loaded, modified and stored in the VisIt *Expressions* window. The additionally specified quantities are given in table 5.4.

Tab. 5.4: Additional field data specified in VisIt

1	2	3	4	5	6
var.	**VisIt var.**	**loc.**	**dim.**	**avail.**	**recomm. disp. mode**
θ	entropy	c	1	B,C,D	Pseudocol., Cont.
v_1	vel_x (2D)	c	1	B,C,D	Pseudocolor
v_2	vel_y (2D)	c	1	B,C,D	Pseudocolor
$\overline{\rho v_1}$	mom_x (2D)	c	1	B,C,D	Pseudocolor
$\overline{\rho v_2}$	mom_y (2D)	c	1	B,C,D	Pseudocolor
$\frac{\partial v_2}{\partial x_1} - \frac{\partial v_1}{\partial x_2}$	vorticity_mag. (2D)	c	1	B,C,D	Pseudocol., Cont.
$\nabla \times \boldsymbol{v}$	vorticity (3D)	c	3	B,C,D	Vector
$\lvert\nabla \times \boldsymbol{v}\rvert$	vorticity_mag. (3D)	c	1	B,C,D	Pseudocol., Cont.
$\frac{\partial p^{(2)}}{\partial x_1}$	dpdx	c	1	B,C,D	Pseudocolor
$\frac{\partial p^{(2)}}{\partial x_2}$	dpdy	c	1	B,C,D	Pseudocolor
$\frac{\overline{P\varphi}}{\overline{P}}$	phase indicator	c	1	D	Pseudocolor

Further, in debug-mode, i.e. if the compiler flag `DEBUG` has been specified before code compilation in `Makefile.para`, the additional quantities in table 5.5 are written during code run, where $\delta\pi$, $\delta p^{(2)}$ and Ψ are the increments resulting from the first projection, the second projection and the volume-of-fluid correction.

After display mode and quantity are chosen in VisIt, clicking the *Draw* button will add the illustration of the selected field quantity to the plotting window. The visualization properties can be modified by clicking on the respective quantity / display mode in the list of added items. More details on VisIt usage are available from the VisIt user's guide[9].

[9] `https://wci.llnl.gov/simulation/computer-codes/visit/manuals` [last visited on August 04^{th}, 2014]

Tab. 5.5: Additional field data in debug mode for visualization using VisIt

1	2	3	4	5	6
Var.	**VisIt var.**	**loc.**	**dim.**	**avail.**	**display mode**
-	cell indices	c	1	A,B,C,D	Label
-	OutputCellDummy0_int_scalar	c	1	A,B,C,D	Pseudocolor
-	OutputCellDummy1_double_scalar	c	1	A,B,C,D	Pseudocolor
-	OutputCellDummy2_double_vector	c	2,3	A,B,C,D	Vector
$\delta\pi$	OutputCellDummy3_cell_increment	c	1	B,C,D	Pseudocolor
Ψ	OutputCellDummy4_vof_increment_cc	c	1	D	Pseudocolor
-	OutputNodeDummy1_double_scalar	n	1	A,B,C,D	Pseudocolor
$\delta p^{(2)}$	OutputNodeDummy1_node_increment	n	1	B,C,D	Pseudocolor

5.4.2 Global Data

Global data can be used for monitoring during and/or after code run. The global data files

1. *globalQuantitiesTimeSeries.inf*
2. *timeSteps.inf*
3. *initialPressureErrorMeasure.inf*

are written while the code is running, information is appended after each time step and their content can be visualized using MATLAB® or GNU Octave in graphics mode by executing the respective .m-file scripts

1. *GlobalQuantitiesMonitor.m*
2. *TimeStepMonitor.m*
3. *InitialPressureErrorMonitor.m*

which can be found in the *Monitoring* folder in the code main directory.

5.4.2.1 *GlobalQuantitiesMonitor.m* for reading *globalQuantitiesTimeSeries.inf*

The *GlobalQuantitiesMonitor* can be used to monitor several global quantities while the code is running of after code execution has been finished. As providing data for monitoring of global quantities consumes a certain amount of computational time, the boolean `monitor_global_quantities` flag in the input file (see XXXXXXX) allows for switching this option on (`TRUE`) an off (`FALSE`). If monitoring is switched on, double precision information is appended to the file *globalQuantitiesTimeSeries.inf* in each time step independent from the field data output interval. The plots displayed after execution of the *GlobalQuantitiesMonitor* are updated each time the "Enter" key is pressed while the *GlobalQuantitiesMonitor* is running and "Q" followed by the "Enter" key will shut down the plots and terminate the *GlobalQuantitiesMonitor*. The information provided depends on the chosen code mode according to table 5.1 and availability of the respective information as shown in the fifth column of table 5.6. The fourth column shows if global sum (tot.), global maximum (max.) or global average (av.) is determined.

Tab. 5.6: Global data for code monitoring at time level $n+1$; *fourth column*: $\otimes$: exactly conserved, $\oslash$: discretely conserved, $\ominus$: not conserved; *fifth column*; code modes A,B,C and D are specified in table 5.1

	expression	**description**	**type**	**avail.**
1	$\sum_{\boldsymbol{c}} (\overline{\rho}\, \Delta V)_{\boldsymbol{c}}$	mass	tot.$^{\otimes}$	B,C,D
2	$\sum_{\boldsymbol{c}} (\overline{\rho \boldsymbol{v}} \cdot \boldsymbol{e}_d\, \Delta V)_{\boldsymbol{c}}$	momentum, coord. dir. d	tot.$^{\otimes}$	B,C,D
3	$\frac{1}{2} \sum_{\boldsymbol{c}} ((\overline{\rho \boldsymbol{v}} \cdot \overline{\rho \boldsymbol{v}})\, \Delta V / \overline{\rho})_{\boldsymbol{c}}$	kinetic energy	tot.$^{\ominus}$	B,C,D
4	$\sum_{\boldsymbol{c}} \left(\lvert\lvert \nabla \boldsymbol{v} \rvert\rvert^2_{Frobenius}\, \Delta V \right)_{\boldsymbol{c}}$	enstrophy[10]	tot.$^{\ominus}$	B,C,D
5	$\sum_{\boldsymbol{c}} (\boldsymbol{\omega})_{\boldsymbol{c}}$	vorticity[11]	tot.$^{\ominus}$	B,C,D
6	$\max_{\boldsymbol{c}} (\boldsymbol{\omega})_{\boldsymbol{c}}$	vorticity[12]	max.	B,C,D
7	$\sum_{\boldsymbol{c}} \left(\overline{P}\, \Delta V \right)_{\boldsymbol{c}}$	XXX enthalpie XXX	tot.$^{\otimes}$	B,C,D
8	$\sum_{\boldsymbol{c}} \left(\overline{P \varphi}\, \Delta V \right)_{\boldsymbol{c}}$	volume-of-fluid (VoF)	tot.$^{\otimes}$	D
9	$\sum_{\boldsymbol{c}} \left(\overline{P}^{(+)} \varphi^{(+)}\, \overline{\varphi}^{(+)}_{rel} \Delta V \right)_{\boldsymbol{c}}$	VoF in ref. phase $(+)$[13]	tot.$^{\otimes}$	D
10	$\sum_{\boldsymbol{c}} \left(\overline{P}^{(+)} \varphi^{(+)}\, \alpha \Delta V \right)_{\boldsymbol{c}}$	VoF in ref. phase $(+)$	tot.$^{\oslash}$	D
11	$\sum_{\boldsymbol{c}} \left(\overline{\rho}^{(+)}\, \overline{\varphi}^{(+)}_{rel} \Delta V \right)_{\boldsymbol{c}}$	mass of ref. phase $(+)$[14]	tot.$^{\otimes}$	D
12	$\sum_{\boldsymbol{c}} \left(\overline{\rho}^{(+)}\, \alpha \Delta V \right)_{\boldsymbol{c}}$	mass of ref. phase $(+)$	tot.$^{\oslash}$	D
13	$\sum_{\boldsymbol{c}} \left(\overline{\varphi}^{(+)}_{rel} \Delta V \right)_{\boldsymbol{c}}$	volume of ref. phase $(+)$[15]	tot.$^{\otimes}$	D
14	$\sum_{\boldsymbol{c}} (\alpha \Delta V)_{\boldsymbol{c}}$	volume of ref. phase $(+)$[16]	tot.$^{\oslash}$	A,D
15	$\sum_{cc} (\Delta A_\Gamma)_{cc}$	interface surface area	tot.	A,C,D
16	$\sum_{cc} (XXXX \Delta A_\Gamma)_{cc}$	interface curvature	tot.	A,C,D
17	$\max_{XXXX} (XXXX)$	interface curvature	max.	A,C,D
18	$d_H \left(\boldsymbol{X}^{(n)}, \boldsymbol{X}^{(n+1)} \right)$	Hausdorff distance[17]	max.	A,C,D

[10] The entstrophy is defined as $\int_\Omega \lvert\lvert \nabla \boldsymbol{v} \rvert\rvert^2_{Frobenius}\, \mathrm{d}V$ in which the squared Frobenius norm is $\lvert\lvert \nabla \boldsymbol{v} \rvert\rvert^2_{Frobenius} = \sum_d \left[\left[\left((\nabla \boldsymbol{v})^T \nabla \boldsymbol{v} \right) \cdot \boldsymbol{e}_d \right]^T \cdot \boldsymbol{e}_d \right]$ and d is the index of the respective coordinate direction.

[11] $\boldsymbol{\omega} = \int_{\partial \Omega_{\boldsymbol{c}}} (\boldsymbol{\nabla} \times \boldsymbol{v}) \cdot \boldsymbol{n}\, \mathrm{d}A$ reduces to $\boldsymbol{\omega}_{2D} = \left(\frac{\partial v_2}{\partial x_1} - \frac{\partial v_1}{\partial x_2} \right)$ in 2 space dimensions

[12] see footnote 11

[13] relative phase indicator of reference fluid phase $(+)$: $\overline{\varphi}^{(+)}_{rel} := \left| \overline{\varphi}_{(VoF)} - \varphi^{(-)} \right| / [\![\varphi]\!]$

[14] see footnote 13

[15] see footnote 13 and footnote 16

[16] only conserved for flows with $\nabla \cdot \boldsymbol{v} = 0$

[17] Hausdorff distance between set of intersections $\boldsymbol{X}$ between interface Γ and grid cell edges at old time level n and at new time level $n+1$: $d_H \left(\boldsymbol{X}^{(n)}, \boldsymbol{X}^{(n+1)} \right) = \max \left(\tilde{d}_H \left[\boldsymbol{X}^{(n)}, \boldsymbol{X}^{(n+1)} \right), \tilde{d}_H \left(\boldsymbol{X}^{(n+1)}, \boldsymbol{X}^{(n)} \right) \right]$ with $\tilde{d}_H (\boldsymbol{Y}, \boldsymbol{Z}) = \sup_{\boldsymbol{y} \in \boldsymbol{Y}} \inf_{\boldsymbol{z} \in \boldsymbol{Z}} d(\boldsymbol{y}, \boldsymbol{z})$ and $d(\boldsymbol{y}, \boldsymbol{z})$ as the distance between the spatial points given by the vectors $\boldsymbol{y}$ and $\boldsymbol{z}$

The file *globalQuantitiesTimeSeries.inf* is a semicolon (;) separated ASCII file with one line for each time step. The information such a line consists of depends on the chosen code mode as stated above. The file header gives the (code mode dependent) number of monitored quantities and lists the information a line contains in the respective executed mode:

```
number of monitored quantities :  <number>
time step number ; time ; Quantity 1 ; Quantity 2 ; ...
; Quantity <number> ; field data output flag ;
<time step number> ; <time>; <Quantity 1> ; <Quantity 2>
; ...  ; <Quantity <number> > ; <field data output flag>
;
```

Both absolute data as stored in *globalQuantitiesTimeSeries.inf* and relative deviation with respect to the corresponding initial value are plotted. The plot of the latter is extended by its time series average and the maximum deviations both from this average and from the initial relative error (which is zero) on the other hand.

5.4.2.2 *TimeStepMonitor.m* for reading *timeSteps.inf*

The *TimeStepMonitor* plots the global maximum absolute value of the fluid velocity over the time step number on the one hand and the time step size over the time step number on the other hand. In the time step size plot an additional marker is displayed for each time step number (also the ones without field data output) indicating which phenomenon has been responsible for the time step size restriction in the explicit part of the respective time step. It is distinguished between

- global maximum velocity (0%)
- global maximum acceleration (XX%)
- viscosity (XX%)
- XXXXX ... (XX%)
- prescribed maximum time step size (XX%)
- temporal distance to next data output level[18] (XX%)

[18] chosen if the temporal distance to the next output time level is smaller than the allowable time step size due to physical/other numerical reasons

- modified time step size due to distance to next data output level[19] (100%)

as source of the current time step size restriction. The percentage values in brackets indicate where the respective markers are plotted between 0 and the maximum on the vertical axis. Further, time step numbers with field data output are marked as well.

The file *timeSteps.inf* is a semicolon (;) separated ASCII file with one line for each time step number. Such a line contains the following information:

```
time step number ; time ; restriction marker ; global
maximum velocity absolute value ; time step size ; field
data output flag ;
```

5.4.2.3 *InitialPressureErrorMonitor.m* for reading *initialPressureErrorMeasure.inf*

With the *InitialPressureErrorMonitor* the behavior of a chosen measure of the nodal pressure $p^{(2)}$ can be monitored during the initial iteration procedure for obtaining an initial pressure that matches the initial conditions of the other quantities. The measure can be chosen to be

- the control volume weighted L_1 norm $\frac{1}{|\Omega|}\sum_{\boldsymbol{c}}\left(|\tilde{p}^{(2)}|\Delta V\right)_{\boldsymbol{c}}$
- the control volume weighted L_2 norm $\sqrt{\frac{1}{|\Omega|}\sum_{\boldsymbol{c}}\left(\left(\tilde{p}^{(2)}\right)^2\Delta V\right)_{\boldsymbol{c}}}$
- the control volume weighted L_∞ norm $\frac{\Delta V}{|\Omega|}\max_{\boldsymbol{c}}\left(\tilde{p}^{(2)}_{\boldsymbol{c}}\right)$

of all nodal pressure values in the computational domain interpolated to grid cell centers second order accurate, $\tilde{p}^{(2)}$. The weights are the grid cell volumes $\Delta V_{\boldsymbol{c}}$ of the respective grid cells $\boldsymbol{c}$.

The file *initialPressureErrorMeassure.inf* is a semicolon (;) separated ASCII file with one line for each iteration step. Such a line consists of the following information:

```
iteration number ; relative change tolerance ; absolute
pressure measure ; relative change of pressure measure ;
```

XXXXXXXXXX

[19] chosen if the temporal distance to the next output time level is slightly larger than the allowable time step size due to physical/other numerical reasons: in this case the allowable time step size is halted in order to avoid a tiny subsequent time step size

SAMRAI[20]

XXXXXXXXXXX

5.5 Initial Conditions

XXXXXXXXXXXXXX

[20] useful documentation can be found under
`http://ktchu.serendipityresearch.org/download/publications/education/SAMRAI_Primer.pdf`
[last visited on Aug 13^{th}, 2014]
`https://computation-rnd.llnl.gov/SAMRAI/download/SAMRAI-Concepts_SoftwareDesign.pdf`
[last visited on Aug 13^{th}, 2014]

6. CONVERGENCE TEST TOOLKIT (CTT)

The Convergence Test Toolkit (CTT) is a collection of bash shell scripts and small SAMRAI based C++ programs inspired by the flow solver STAFSEA^2 with kernels for different subtasks in order to determine convergence behavior of any compiled version of STAFSEA^2. The framework is located in

```
<STAFSEA² main folder>/Utilities/CTT
```

and once executed by calling

```
./CTTstart <restart data>
```

via the command line within the directory given in the first box above it will generate a STAFSEA^2 executable (if not already available) and run it on a hierarchy of grids as shown in Fig. 6.1, followed by automatic comparison of solutions on grids of different resolution, computation of error norms and determination of convergence orders. In the above (and following) boxes

- <*STAFSEA*^{2} *main folder*> represents the STAFSEA^2 main directory containing files like `main.C`, `mainFunc.C` and `Makefile.para` as well as folders like `FlowSolverBasics`, `Predictor` and `Corrector`
- <*restart data*> represents the name (not path !) of the directory in which STAFSEA^2 stores double precision restart data, typically this is `RestartData`. If no input argument is specified this default name is assumed.

The MATLAB®/Octave script

```
convOrdViz.m
```

in the `CTT` folder can be used to visualize the results stored in the file

```
convergenceOrder.dat
```

which is produced based on the results stored in

```
errorNorms.dat
```

considering the auxiliary file

```
variables.inf
```

after grid matching and error norm computation. In order to perform a successful convergence test using the CTT, the following steps have to be followed:

1. go to <*STAFSEA*^{2} *main folder*> in the command line and type

   ```
   make clean
   ```

   followed by `enter` in order to remove the existing STAFSEA^2 executable.

2. configure the STAFSEA^2 version subject to the subsequent analysis via the compile time parameter file `Makefile.para` in <*STAFSEA*^{2} *main folder*> as described in section 5.1.

   Note, that proper convergence results will only be obtained with limiters switched off by setting `-DLIM=0` (resulting in unlimited central slopes).

3. set up the problem (initial conditions, boundary conditions, constants ...), which should be computed for convergence test, via the subsequently used input file in <*STAFSEA*^{2} *main folder*>`/InputFiles` as described in section 5.2.

   Note, that several input file parameters (like time step size, grid resolution, load balancing etc.) are adjusted automatically by the CTT after execution.

4. set convergence test parameters using the file

   ```
   ./ParaInput.usr
   ```

   in the CTT folder as described in section 6.1.

5. call the script

   ```
   ./reset
   ```

   within the CTT folder in order to remove compiled executables for the subtasks and already existing output and log files.

6. execute `CTTstart` as given on page 31. Details on the script and its subscripts and -programs are given in section 6.2. Steps 5. and 6. are condensed in the script

   ```
   ./newrun
   ```

   assuming `RestartData` to be the name of the solver's double precision restart data folder.

7. wait until the file `convergenceOrder.dat` is successfully written. The layout of the screen output produced by the `CTT` is adopted from the Solver Test Environment (`STE`), see section 7.

8. visualize results via `convOrdViz.m` as described in section 6.4.

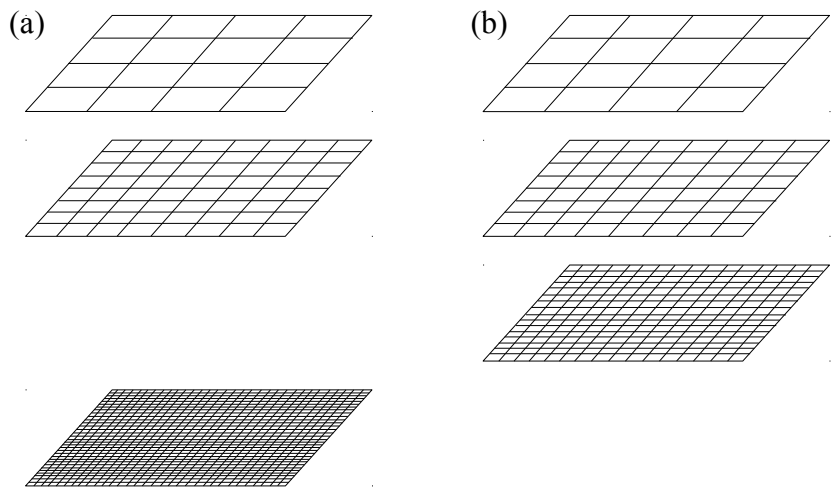

Fig. 6.1: Hierarchy of grids (a) with and (b) without fine grid for reference solution

6.1 Convergence Test Configuration

The convergence test process has to be configured before execution via the file

```
./ParaInput.usr
```

in the `CTT` directory. There are 14 parameters to be specified. Note, that all lines in this files – besides of the ones containing the 14 parameters – need to begin with # in order to being skipped by the file parser[1]. The parameters in `ParaInput.usr` are

[1] see 6. on page 37

1. (string) absolute path of <*STAFSEA*[2] *main folder*>
2. (string) relative path of the STAFSEA[2] input file, which should be used for the analysis, with respect to <*STAFSEA*[2] *main folder*> specified in 1.
3. (integer) number of available processors
4. (integer) number of spatial dimensions[2]
5. (integer) number of sample points in the resulting convergence plots[3]

   Note, that N sample points as shown in Fig. XXXXX require computation of numerical solutions on $N+1$ computational grids, if an exact solution is available in addition, and on $N+2$ computational grids (with refinement factor of 2 in each spatial direction between any grid and its subsequent one), if no exact solution is available and self-convergence of the numerical method is tested or a fine grid reference solution is substituting an exact solution instead. At least 2 sample points[4] are required in order to determine convergence behavior of the numerical method STAFSEA[2] is compiled for.
6. (integer) number of grid cells in each spatial direction on the coarsest grid (blank key separated)

   Note, that

   - with N sample points[5] the number of grid cells in each spatial direction on the finest grid is 2^N times the one on the coarsest grid if an exact solution is available and (at least)[6] 2^{N+1} if no exact solution is available.
   - the number of grid cells per direction on the coarsest grid, divided by the number of used processors[7], has to be an integer number $\geqslant 4$ due to the way domain decomposition is handled during the analysis.
   - while choosing the number of grid cells on the coarsest grid too large will result in very expensive computations on the finer grids, choosing the number of grid cells on the coarsest grid too small might lead to inaccurate convergence results due to under-resolved settings.

[2] 2 or 3; currently analysis only works for 2 spatial dimensions
[3] see section 6.4
[4] ideally 4 or more sample points should be used, which, however, significantly increases computational time and computations of the finest grids required might not fit in the storage of the machine used
[5] see 5. on page 34
[6] see 13. on page 35
[7] see 3. on page 34

7. (integer) number of time steps on the coarsest grid

   Note, that with N sample points[8] the number of time steps on the finest grid is 2^N times the one on the coarsest grid if an exact solution is available and (at least)[9] 2^{N+1} if no exact solution is available (with 2^N / (at least) 2^{N+1} times the number of grid cells in each spatial direction[10] and, thus, 2^N / (at least) 2^{N+1} times the cost per single time step compared to the one on the coarsest grid) due to the CFL condition for computation with the same constant CFL number[11] on each grid[12].

8. (float) CFL number (on the finest grid)[13] with upper limit of 0.5 (for two space dimensions)

9. (float) maximum allowable time step size[14]

10. (float) refinement factor r

11. (integer) parameter for switching determination of the numerical solutions on the hierarchy of grids either on (`1`) or off (`0`). The latter can be chosen if the solutions already exist and only the analysis has to be performed. If determination of numerical solutions - on the other hand - is switched on (`1`), existing solutions in folders `Solution`$\star$ – with $\star$ indicating arbitrary extensions to `Solution` – are deleted.

12. (integer) switch to choose between either the same constant CFL number on all grids (0), which is the mode to choose for a proper convergence test in both space and time, and constant number of time steps for each grid (1). The latter implies, that the CFL number specified[15] is only applied on the finest of all grids used while solutions on all coarser grids are computed with reduced CFL numbers to guarantee for the same number of time steps on each grid.

13. (integer) switch $S \in \mathbb{Z}$ to choose between three reference solution modes[16]:

[8] see 5. on page 34
[9] see 13. on page 35
[10] see 6. on page 34
[11] see 8. on page 35
[12] see 12. on page 35 and Fig 6.1
[13] see 12. on page 35 and Fig 6.1
[14] set the maximum allowable time step size to a high double value in order to ignore this upper limit and let `CTT` determine the maximum allowable time step size automatically
[15] see 8. on page 35
[16] for more details see section 16. on page 43

- $S > 0$: a **numerical solution on a reference grid**, which is much finer than the finest grid in the hierarchy of grids, is used instead of an exact solution as shown in Fig. 6.1 (a).
- $S = 0$: one more solution within the grid hierarchy is computed as shown in Fig. 6.1 (b) instead of a solution on a much finer reference grid in absence of an exact solution. In this case, numerical solutions on 3 different grids are needed for obtaining one sample point[17] in the resulting convergence plot and self-convergence is tested instead.
- $S < 0$: An **exact solution** is used to compare the numerical solutions with. In this case $|S|$ is a flag for choosing a specific exact solution suitable to the problem setup. This is not implemented yet.

14. (float) refinement factor between finest grid in the grid hierarchy and grid of numerical reference solution if the switch in 13. is set to $S > 0$. The refinement factor between finest grid level of the grid hierarchy and reference grid has to be 2^k with $2 \leqslant k \in \mathbb{N}$. For $S \leqslant 0$ this is ignored.

6.2 CTT Description

The script `CTTstart` controls the convergence test process and calls the different scripts

- `bashcalc.sh`
- `SCRIPTadjustInputFile`
- `SCRIPTcodeFunctionManagement`
- `SCRIPTseekAndDestroy`
- `SCRIPTstoreDirInfo`

and `SAMRAI` based C++ programs

- `checkInput`
- `convCheck`

[17] see 5. on page 34

- errorNorms
- iniTimeStep
- matchGrids

for the different sub-tasks after sourcing the file

```
functions.sh
```

which includes the STE[18] file

```
STE/includes/log.sh
```

for visualization routines and provides several functions used while CTTstart is running. The C++ program executables listed above are, if not already available in the CTT directory, compiled during execution of CTTstart. The latter involves **the following steps**, which **are** all **processed automatically** after calling

```
./CTTstart <restart data>
```

in the command line, so non of them needs to be done manually:

1. check for availability of STE
2. sourcing functions.sh
3. enabling the shell based calculator bashcalc.sh[19]
4. specification of some file names and constants
5. check of input arguments of CTTstart:
   - name of the folder the flow solver STAFSEA[2] stores double precision restart data in (typically RestartData). if nothing is specified, RestartData is assumed
6. check and reorganization of the user input from ParaInput.usr. This task is done by the C++ program checkInput and consists of the following steps:
   - check if executable checkInput already exists.
   - if checkInput does not exist, it is compiled based on the files

[18] see section 7
[19] bash shell calculator bc with 16 digit output

  - `Makefile_checkInput`
  - `checkUserInput.C`
  - `checkUserInput.h`
- `checkInput` is executed with input arguments
  - parameter input file name (`ParaInput.usr`)
  - name of parameter output file after input check for further internal use within `CTTstart` (`para.input`)

7. specification of some paths and file names based on user input

8. preparation of the CTT directory. This involves removal of old results in `*.dat` and `*.inf` files using the script `SCRIPTseekAndDestroy` with the path of the respective file to remove as input argument, and, if new numerical solutions have to be determined (see 11. on page 35), removal of all folders containing already existing solutions in `Solution*` directories where `*` indicates arbitrary extensions to `Solution` as well as gathering of the flow solver parameter file `Makefile.para`. Further, in this case no numerical solution needs to be computed as analysis is subject to already existing data the latter is checked for consistency

9. check if a single phase or a two phase problem/code version has to be analysed and store information to the file

   ```
   phases.inf
   ```

   which will be stored in the folder of each solution level

10. getting copies of functions from `STAFSEA`^{2} which are utilized by the `CTT` C++ sub-programs using the script `SCRIPTcodeFunctionManagement` with these input arguments:

    - flag for switching between getting (1) and removing (0) copies of functions from `STAFSEA`^{2} into/from (1/0) the `CTT` directory
    - number of fluid phases (1 or 2) as the two phase analysis requires more `STAFSEA`^{2} functions
    - path to the `STAFSEA`^{2} main directory

11. getting the `STAFSEA`^{2} executable subject to the convergence test. This involves these steps:

- check if executable `STAFSEAsquared` already exists
- if `STAFSEAsquared` does not yet exist, it is built based on parameters specified in `Makefile.para` in *<STAFSEA² main folder>* (see page 31)
- removal of auxiliary files – if existing – used for detection of successful flow solver termination (`done.inf`)

12. determination of the allowable time step size on the coarsest of all grids if the set of numerical solutions needs to be determined prior to analysis. This involves

    - adjustment of the `STAFSEA`^{2} input file specified in `ParaInput.usr` via the script `SCRIPTadjustInputFile` which parses the given input file to a modified input file subject to the arguments
      - number of available processors
      - path of `STAFSEA`^{2} input file
      - path of modified target input file
      - CFL number (specified/pre-computed)[20]
      - final time of computation
      - pre-computed number of time steps
      - pre-computed initial maximum allowable time step size
      - pre-computed number of grid cells in each spatial direction (blank key separated)
    - execution of the flow solver executable `STAFSEAsquared` on the coarsest grid for (at least) the number of time steps specified for this grid in order to determine the minimal occurring time step size during this computation if time step size is not restricted. This is done in order to guarantee for stable subsequent computations by enforcing the used time step sizes to be fractions of the allowable time step size on the coarsest grid.
    - determination and storage of the determined allowable time step size on the coarsest grid via the C++ program `iniTimeStep` consisting of the following steps:
      - check if `iniTimeStep` already exists
      - if `iniTimeStep` does not exist it is compiled based on the files

[20] see 12. on page 35

        * `Makefile.para`
        * `Makefile_iniTimeStep`
        * `iniTimeStep.C`
        * `iniTimeStep.h`
      - `iniTimeStep` is executed with input arguments
        * path of the file where the time step sizes of all time steps of the previous run of `STAFSEAsquared` are stored (typically `./timeSteps.inf`)
        * path of the file in which the determined allowable time step size is stored (`./iniTimeStep.inf`)
      - check if execution of `iniTimeStep` has terminated successfully before continuing
        The latter searches for existence of the file `done.inf` which is generated by each of the C++ sub-programs upon successful termination
  - reducing the determined time step size by multiplication with 0.9 as a safety factor concerning stability.

13. computation of the finally applied mode-depending[21]
    - final computation time
    - number of time steps
    - fix time step size

    on the coarsest grid.

14. determination of numerical solutions on the hierarchy of grids. This involves
    - creation of a directory for storage of the numerical solution
    - computation of the parameters which are specific for each grid and of the resolution of the latter (number of grid cells, time step size, number of time steps ....)
    - adjustment of the `STAFSEA`^{2} input file according to the determined grid resolution specific parameters. The script `SCRIPTadjustInputFile` therefore parses the given input file to a modified input file subject to the arguments
      - number of available processors

[21] see 12. on page 35

- path of `STAFSEA`^{2} input file
- path of modified target input file
- CFL number (specified/pre-computed)[22]
- final time of computation
- pre-computed number of time steps
- pre-computed initial maximum allowable time step size
- pre-computed number of grid cells in each spatial direction (blank key separated)

A equal box/patch distribution on each grid refinement level is enforced by adjustment of the `STAFSEA`^{2} input file parameters

- `smallest_patch_size`
- `largest_patch_size`

of `level_0` of the adaptive mesh refinement (AMR) hierarchy of each grid, such that both patch size limits are equal. This only works with a single grid resolution level per computation and therefore also the parameter `max_levels=1` is set. The parameters in the modified `STAFSEA`^{2} input files, changed automatically by `SCRIPTadjustInputFile`, can be identified due to some leading blank keys.

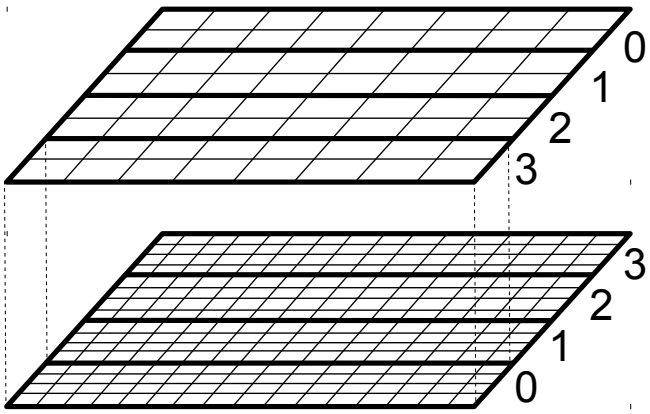

Fig. 6.2: Matching of solutions on congruent patches (fat) of grids of different resolution based on patch coordinates (not patch numbers); here a refinement factor $r = 2$ is used between the two grids per coordinate direction and each domain is divided into 4 subdomains (boxes/patches) 0-3 according to 4 processors used for parallel solution generation. The box/patch layout is identical on each grid.

[22] see 12. on page 35

In the subsequent analysis[23], congruent boxes/patches from the different computations are detected coordinate based as the box/patch numbering in general differs from computation to computation and solutions of different resolution are matched[24] on the respective congruent patch combination.

- removal of old existing `STAFSEA`^{2} output
- parallel execution of the flow solver executable `STAFSEAsquared` as described in section 5.3, subject to the respective modified input file and the specified number of processors
- storage of the resulting numerical solution and of additional information on which restart data set (directory) at which time step has to be used in the following analysis. This information is stored by the script `./SCRIPTstoreDirInfo` in a file `dir.inf` which can be found in each solution folder.

15. generation of executables of the remaining `SAMRAI` based C++ programs needed for analysis of the generated numerical solutions on the hierarchy of grids, if not already available:

    `matchGrids` is built based on

    - `Makefile.para`
    - `Makefile_matchGrids*`
    - `matchGrids.C` (main program)
    - `matchGrids.h` (main program header)
    - `match.C` (class)
    - `match.h` (class header)
    - `match_matching.C` (kernel)

    `errorNorms` is built based on

    - `Makefile.para`
    - `Makefile_errorNorms`
    - `errorNorms.C` (main program)

[23] see 16. on page 43

[24] computation of the difference of coarse and fine grid solution in the location associated with the coarse grid solution

- `errorNorms.h` (main program header)
- `norms.C` (class)
- `norms.h` (class header)
- `norms_compute.C` (kernel)

`convCheck` is built based on

- `Makefile.para`
- `Makefile_convCheck`
- `convCheck.C`
- `convCheck.h`

16. convergence test:

    Assuming that

    - the grid hierarchy consists of L grid levels indexed $0 \leqslant n \leqslant L-1$
    - $\overline{\Phi}^{[m]}$ denotes the integral average of the exact solution $\Phi(\boldsymbol{x})$ over a region $\Omega^{[m]}$
    - $\overline{\Phi}^{[m]}_{\mathfrak{h}(n)}$ denotes[25] the integral average over a region $\Omega^{[m]}$ based on integral averages computed in grid cells on grid level n with grid level parameter[26]

      $$\mathfrak{h}(n) := \frac{h}{\prod_{j=0}^{n} r_{(j)}} \tag{6.1}$$

      and level depending grid refinement factor

      $$r_{(n)} \begin{cases} = 1, & n = 0 \\ > 1, & n > 0 \end{cases} \tag{6.2}$$

      with $r_{(n)} \in \mathbb{R}$ as well as grid parameter

      $$h = \begin{cases} c_0 \frac{\max_d (v_d)}{\min_d \left(\Delta x_d^{[0]} \right)} \Delta t^{[0]} = c_0 \, CFL \\ c_d \frac{1}{\min_d \left(\Delta x_d^{[0]} \right)} \Delta x_d^{[0]} \end{cases} , \quad d \in (1, D) \tag{6.3}$$

      of the coarsest grid at level $n = 0$ with $\mathcal{O}(1)$ constants c_j

[25] the lower index $\blacksquare$ represents the grid the data is computed on by the flow solver and the upper index $^{[\square]}$ represents the grid the data is matched to during analysis

[26] grid parameter of the coarsest grid divided by the refinement factor between coarsest grid at level $n = 0$ and current level n

then the numerical solution $\overline{\Phi}^{[n]}_{\mathfrak{h}(n)} = \overline{\Phi}_{\mathfrak{h}(n)}$ on grid level n deviates from the reference solution[27] $\overline{\Phi}^{[n]}_{ex}$ by

$$\begin{aligned}\overline{\Phi}^{[n]}_{\mathfrak{h}(n)} &= \overline{\Phi}^{[n]}_{ex} + \mathcal{O}(\mathfrak{h}^q(n)) \\ &= \overline{\Phi}^{[n]}_{ex} + \mathfrak{h}^q(n)\, C \end{aligned} \tag{6.4}$$

with problem depending $\mathcal{O}(1)$ error constant C. The convergence order q in the p-norm is determined

- according to
$$q_n = \frac{\log R_n}{\log r} \tag{6.5}$$
in case of **the same constant** $r \in \mathbb{N}$ **with** $r > 1$ **for each grid level** n, yielding $\prod_{j=0}^{n} r_{(j)} \equiv r^n$ due to $r_{(0)} = 1$. Equation (6.5) can be applied
  - in case an exact solution or a fine grid reference solution is available, with
$$R_n := \frac{\left\| \overline{\Phi}^{[n]}_{\mathfrak{h}(n)} - \overline{\Phi}^{[n]}_{ex} \right\|_p}{\left\| \overline{\Phi}^{[n+1]}_{\mathfrak{h}(n+1)} - \overline{\Phi}^{[n+1]}_{ex} \right\|_p} \tag{6.6}$$
resulting from elimination of C using equation (6.4) twice on two different levels n
  - if neither an exact solution nor a fine grid reference solution is available and self-convergence is tested instead, with
$$R_n := \frac{\left\| \overline{\Phi}^{[n]}_{\mathfrak{h}(n)} - \overline{\Phi}^{[n]}_{\mathfrak{h}(n+1)} \right\|_p}{\left\| \overline{\Phi}^{[n+1]}_{\mathfrak{h}(n+1)} - \overline{\Phi}^{[n+1]}_{\mathfrak{h}(n+2)} \right\|_p} \tag{6.7}$$
after elimination of C and $\overline{\Phi}^{[n]}_{ex}$.
- according to
$$q_n = \frac{\log R_n}{\log r_{(n+1)}} \tag{6.8}$$
in case of **different** $r_{(n)} \in \mathbb{R}^+ \backslash \mathbb{N}$ **for each grid level** n with R_n according to equation (6.6) since equation (6.8) can be derived only in case of existence of an exact or fine grid reference solution.

[27] integral average of the exact solution, $\overline{\Phi}^{[n]}$, or the numerical solution on a fine reference grid, $\overline{\Phi}^{[n]}_{\mathfrak{h}(k)}$, both labeled $_{ex}$

If neither an exact solution nor a fine grid reference solution is available, the equation

$$\log\left(R_n\right) - \log\left(r_{(n+2)}^{q_n}\right) - \log\left(r_{(n+1)}^{q_n} - 1\right) + \log\left(r_{(n+2)}^{q_n} - 1\right) = 0 \tag{6.9}$$

has to be solved for q_n numerically instead, with R_n according to relation (6.7). For

$$r_{(n+1)} \equiv r_{(n+2)} \tag{6.10}$$

(which implies that all r are identical), equation (6.9) reduces to equation (6.8). The condition (6.10), however, can only be achieved for $r \in \mathbb{N}$ since for the number k_D of grid cells per coordinate direction $k_D \in \mathbb{N}$ has to be satisfied for each grid level n and refinement factor r.

Thus, the analysis consists of the following sub-steps:

(a) `matchGrids` with input arguments

- number of processors
- folder of the numerical solution on the finer grid (or exact solution)
- folder of the numerical solution on the coarser grid
- name of modified input file

reads **solutions on finer grids** of level $n+1$ with $0 \leqslant n \leqslant L-1$ **(and reference or exact solutions)**, transforms the latter to solutions matching the respective coarser grids to guarantee that difference computation according to equations (6.6) or (6.7) is done with integrals over the same spatial region $\Omega_i^{(n)}$ – grid cells of the coarser grid – on both coarser and finer grid (or of the exact solution), and stores the results per box/patch in binary files

`matchedData_patch<patchNo>.bin`

in the solution folder of the coarser grid with <*patchNo*> being a 5 digit integer number with leading zeros (e.g. `00002` for patch #2)[28].

Suitable finer grid (or exact solution) data is obtained via integration over the domain $\Omega_i^{(n)}$ of a coarser grid cell. For

[28] for example, if the difference $\overline{\Phi}_{\mathfrak{h}(n)}^{[n]} - \overline{\Phi}_{\mathfrak{h}(n+1)}^{[n]}$ has to be computed in equation (6.7), solution data from folder `Solution_Level_<n+1>` is read and integrated over grid cells of level n and the resulting averages are stored in folder `Solution_Level_<n>` for subsequent differencing with solutions determined on the coarser gird

- an arbitrary refinement factor $r > 1$ with $r \in \mathbb{R}^+ \backslash \mathbb{N}$, arbitrary fractions of several finer grid cells – labeled (ref) – can contribute to the resulting value

$$\overline{\Phi}^{[n]}_{(ref)} = \int_{\Omega_i^{(n)}} \Phi_{(ref)} \, \mathrm{d}V \tag{6.11}$$

  in a coarser grid cell on level n as shown in Fig. 6.3 (a) if no exact solution is used as reference. For sharp interface two phase flow even in the latter case suitable reference (or exact) data can not be determined without the use of geometry information and state recovery and is therefore not applied.

- a grid refinement factor of $r \in \mathbb{N}$, one coarser grid cell contains complete fine grid cells only as shown in Fig. 6.3 (b). In this case the integration of finer grid data over a coarser grid cell can be simplified to

$$\overline{\Phi}^{[n]}_{(ref)} = \frac{\sum_{i=1}^{r^D} \left(\Delta V_{(ref)} \overline{\Phi}_{(ref)} \right)_i}{\Delta V_{(n)}} \tag{6.12}$$

  with

$$\Delta V_{(n)} := \int_{\Omega_i^{(n)}} \mathrm{d}V \tag{6.13}$$

  and

$$\Delta V_{(ref)} := \int_{\Omega_i^{(ref)}} \mathrm{d}V \tag{6.14}$$

  as the volumes of the coarser and finer grid cell and D as the number of space dimensions

While a refinement factor $1 < r < 2$ can save a quit huge amount of computational time during generation of numerical solutions on the different grids, the choice of $r = 2$ leads to a less complex solution matching process according to equation (6.12) as stated above. Thus, yet only the latter option of $r = 2$ is implemented.

Besides of files `matchedData_patch<patchNo>.bin`, the file

```
patchInfo.dat
```

is written, in which patch numbering and respective lower and upper spatial bounds are stored. Further, the file

```
variables.inf
```

is written storing the order variable names.

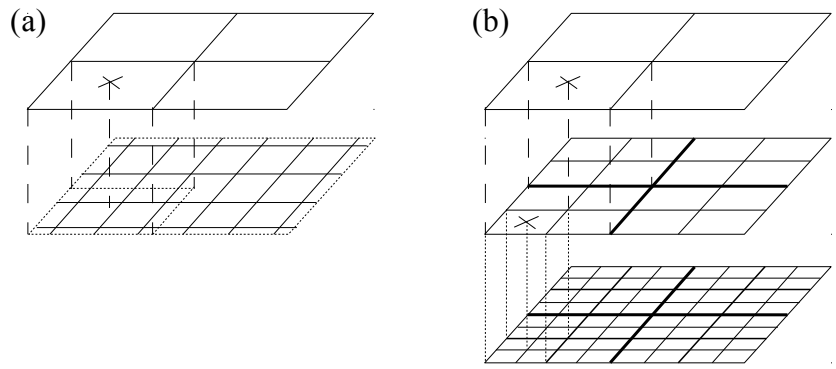

Fig. 6.3: Grids with (a) arbitrary refinement factor $r \in \mathbb{R}^+ \backslash \mathbb{N}$ and (b) $r \in \mathbb{N}$ (here $r = 2$)

(b) `errorNorms` with input arguments

- number of processors
- index of the current solution difference $\Delta\overline{\Phi}$ the norms should be computed from (each difference has the index of the coarser of the pair of grids; in case of usage of an exact solution the index is the one of the grid the numerical solution is computed on)
- directory of the solution with the corresponding index
- name of modified input file

reads **solutions on coarser grids** of level n with $0 \leqslant n \leqslant L-1$ from folders `Solution_Level_<n>`, together with the matched fine grid data from files `matchedData_patch<patchNo>.bin` in the same folder, computes the differences $\Delta\overline{\Phi}$ of numerical solutions $\overline{\Phi}$ as needed in relation (6.6) or (6.7), respectively, and determines the p-norms

$$\|\Delta\overline{\Phi}\|_p = \left(\sum_i \left(\Delta\overline{\Phi}_i\right)^p \right)^{\frac{1}{p}} \tag{6.15}$$

of the determined differences $\Delta\overline{\Phi}$ in each coarser grid cell i with $p \in (1, 2, \infty)$. The ∞-norm corresponds to the $\max$-function. Results are stored in file `errorNorms.dat`.

(c) `convCheck` with input arguments

- refinement factor r
- number of sample points (see 5. on page 34)

finally determines the convergence order q_n from equation (6.5), (6.8) or (6.9), respectively, for each of the

$$N = \begin{cases} L-1, & S = 0 \\ L-2, & S \neq 0 \end{cases} \tag{6.16}$$

sample points and stores results in file `convergenceOrder.dat` which can be visualized using the MATLAB®/Octave script `convOrdViz.m`[29].

6.3 Output Files

Three output files are generated by the CTT:

6.3.1 `errorNorms.dat`

This file contains $N + 1$ data blocks after execution of `errorNorms`, separated by an empty row. Each block in `errorNorms.dat` is headed by the grid spacing of the coarsest grid involved. All other lines of each block contain the computed p-error-norms of the differences $\Delta\Phi$ with $p \in (\infty, 1, 2)$ from left to right, separated by colons with preceding and following blank keys. Each of these lines represents one physical quantity.

6.3.2 `convergenceOrder.dat`

This file contains N data blocks after execution of `convOrd`, separated by an empty row. Each block is headed by the sample identifier $0 \leqslant l \leqslant N - 1$. All other lines of each block contain the computed convergence rates corresponding to the p-error-norms in section 6.3.1 with $p \in (\infty, 1, 2)$ from left to right, separated by colons with preceding and following blank keys. Each of these lines represents one physical quantity.

6.3.3 `variables.inf`

This file is written during the run of `matchGrids` and contains the names of the variables subject to convergence test.

[29] see section 6.4

6.4 Visualization

The following visualization options are available after successful run of the CTT:

- visualization of both error over grid resolution and convergence order in L_1, L_2 and L_∞ norms using the MATLAB®/Octave script `convOrdViz.m` or, alternatively, the python script `convOrdViz.py`
- visualization of error distribution and location of maximum occurring error using VisIt

6.4.1 Visualization using the `convOrdViz` script

The MATLAB®/Octave script `convOrdViz.m` or, alternatively, the python script `convOrdViz.py` reads the files

- `convergenceOrder.dat`
- `errorNorms.dat`
- `variables.inf`

and provides graphical output of the results of both error over grid resolution (from `errorNorms.dat`) and convergence order (from `convergenceOrder.dat`) in L_1, L_2 and L_∞ norms. The additional file `variables.inf` is needed in order to assign each line of each block of results in the respective `.dat` file to a variable name. The `variables.inf` file was generated during solution matching (`matchGrids`).

6.4.2 Visualization of spatial error distribution and location of maximum error using VisIt

The folder `ErrorWithRespectToFinerGridLevel_VisIt` is created during computation of `errorNorms` in the `Solution` folders of all but the finest grid level, containing VisIt data of the local differences occurring in equations (6.6) and (6.7) on the coarser of both involved grids at final time. The data is governed by the

`dumps.visit` file in this folder[30]. After loading this file in VisIt, for each quantity subject to the convergence test both spatial error distribution (`_error`) and location of maximum occurring error per patch[31] (`_maxErrorLocation`) are available for visualization and can be displayed using the VisIt Pseudocolor plot after choosing the respective quantity.

6.5 CTT Limitations

Currently, the `CTT` is subject to the following restrictions:

- upper analysis bound is a convergence order of $q_n = 2$ due to interpolation and integration during the matching process of solutions on different grids
- if `index_data` are used by the flow solver (e.g. in context with a leve-set), numerical solution generation is limited to 1 processor only as the analysis is performed sequentially and the `SAMRAI restart-redistribute` script, which converts parallel data into sequential data in the applied case, does not support `index_data` redistribution.
- yet only refinement factor of $r = 2$ (or $r \in \mathbb{N}$, respectively) is possible
- no inclusion of exact solution is possible yet
- load balancing is switched of during parallel solution determination in order to guarantee for a prescribed box/patch configuration for the analysis which increases cost of the computation of the numerical solutions
- yet limited to two space dimensions

[30] the `dumps.visit` file in each `Solution` folder belongs to the numerical solution, the `dumps.visit` file in each `ErrorWithRespectToFinerGridLevel_VisIt` folder belongs to the convergence analysis results (error distribution and max. error location per patch).

[31] the maximum error location value and coloring represents the patch local 1-dimensional cell index

7. SOLVER TEST ENVIRONMENT (STE)

XXXXXX